2/9/33 (Item 11 from file: 47)
DIALOG(R)File 47:Gale Group Magazine DB(TM)

02731668 SUPPLIER NUMBER: 03896010 (THIS IS THE FULL TEXT)
Lesson one: durable, doable databases. (Process-Driven Data Design, part 1)
Sweet, Frank
Datamation, v31, p83(2)
Aug 15, 1985
CODEN: DTMNA LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT
WORD COUNT: 1213 LINE COUNT: 00094

TEXT:

Have you ever noticed that there are just two kinds of data entities in the real world? Imagine, say, a 10,000-record **vendor file** where, on average, 100 **new vendors** are **added** each month and 100 inactive ones purged. Its monthly volatility (turnover divided by population) is 1%. Now consider a file of 1,000 purchase orders where 1,000 orders are added each month and the same number closed. Its volatility would be 100%.

Compute the volatility of the permanent files in your dp shop and you'll uncover something strange; there are two populations out there--two classes of records. A histogram shows two peaks: one around 1% like our vendor example, the other near 100% like the purchase orders. Volatility is binary, it seems, and almost every record falls into one group or the other. Why is this? Does it apply to every database? Does it reveal an important underlying truth about data?

The answer to the last two questions is yes. But to find the why, we will journey into the world of database design. We will see that objectively good designs exist, and we'll learn to recognize their traits. We'll review equations in data flow dynamics like the above, and mnemonics like K=DU.sup.3. We'll manipulate the boxes and arrows of Bachman diagrams to derive, split, and merge entities. We'll examine six useful design patterns we can apply to many applications. And we'll examine six useful design patterns we can apply to many applications. And we'll return with a checklist that, applied to a design, tells us if it's ready to be built.

In all, we'll cover 14 topics in the next few months:

* Durable, Doable Databases--the marks of a sound design
* Data Flow Dynamics--the mathematics of data movement
* Objects and Events--the two populations of records
* Keyfield Design--dataless, unique, unambiguous, unchanging
* The Two-Headed Arrow--many-to-many relationships
* The Headless Arrow--one-to-one relationships
* Optional Arrows--splitting entities
* Hard Sets, Soft Sets, and Summary Fields--integrity vs. ease of use
* The Identifier and Its Name--the most-used design pattern
* The Past Event--when audit trail is mandatory
* The Future Event--productive skepticism
* The Template--cookie-cutter records
* The Self-Related Record--bill-of-material structure
* Numbers, Keyfields, and Real Things--a completion checklist

The hallmarks of a sound database design are that it is durable and doable. A durable design survives changes in the business environment. Managers, policies, products, and styles all come and go. But a database represents millions of dollars' worth of painstakingly collected data about our firm and we'd rather it didn't come and go with them. A doable design is one that's easy to implement. It capitalizes on the one gift we all share (that we get better at anything each time we repeat it) and doesn't penalize our common flaw (we never get something right the first time).

Durable databases can survive the ver applications that created them. I recall a shop that has been through two payroll systems, a now-defunct workmen's comp system, and two security systems in the past six years. Yet

their employee database, which those applications used, has been chugging away uninterrupted since it was first installed. At the other extreme lies the painful case of a design so dependent on transitory management style that it was unknowingly wrecked by the stroke of an executive pen mere weeks after implementation. Understand, durability is no accident. We deliberately build it into designs by modeling underlying business reality and by making sure the result is shared among applications and can be expanded with new data.

REFLECT REAL EVENTS

Modeling reality means that our data structures reflect real events, real activities. They must not simply mimic records in other filing systems such as forms and documents. Forms can disappear without a trace, while products, vendors, customers, and employees cannot. We model reality by letting our designs be determined by the business processes they will support. We call this process-driven data design, and we will return to how it is done in a moment.

Sharing it among applications means that different people use the same database for different purposes. Avoiding redundancy, DBAS call this, and there are three reasons why it's important. First, redundant data implies redundant data updators, and it's pointless to have two people doing the same job. Also, the more users, the greater the incentive to keep it accurate, which means the data are more timely and reliable for everyone. Finally, the more widely a database is shared, the more durable it is. Its users hold it steady despite our yearly reorganizational tempests.

Making it expandable so new data can be added means developing the skills and tools to add new fields to existing records, new records to existing files, or new files to the database without shutting down or retrofitting existing applications. Expandable databases are more durable because each new application brings new data needs. Application developers will use the central database if their needs are easily accomodted. Otherwise, each group could go its own way, and the centrally shared data pool would be stillborn.

The most doable database designs are those that can be brought up faster and with less effort than comparable flat files. The worst are disasters where the application quickly reaches 90% complete and stays there for months. There are two secrets to making it doable: include only what we need, and keep it simple.

Data design conceals a treacherous twist to the dilemma: do we want it right or do we want it Friday? It's the urge to include too much. Designing a vendor file for a payables system, we conclude that we need vendor ID number, name, address, and amount owed. We should stop, but temptation draws us on. We're designing a database to be shared by future applications, we reason. Shouldn't we find out what they'll need and include it too? The bait looks tempting--doing a thorough job--and the risk looks slight--a few days' extra work. But, to do it, we'd have to identify every data element and every vendor attribute that any user could ever want for any conceivable purpose. In short, it becomes an endless undertaking.

The power of database management systems is that they enable records to be stretched to include new fields as new applications arise, and do so without making us track down and recompile exisiting programs. Consequently, the first way we make a database design doable is to include only what we need at the time we design it.

The second secret of doability is to keep it simple, and this implies modularity. There are really only a handful of basic database design patterns. We use them like building blocks, in one application after another. We build designs out of them rather than deriving each application from scratch. We'll introduce these patterns in the ninth installment. We'll start, though, next time with dataflow dynamics.